Clustering de Documentos a partir de Métricas de Similitud basado en Big Data

Laura Mejía Vásquez
Universidad EAFIT
lmejiav6@eafit.edu.co

Daniel Rendón Montaño
Universidad EAFIT
drendon9@eafit.edu.co

Dillan Muñeton Avendaño
Universidad EAFIT
dmuneto1@eafit.edu.co

Juan Fernando Ossa Vásquez
Universidad EAFIT
jossava@eafit.edu.co

17 de noviembre de 2017

Resumen

En este documento se desarrolla una posible solución al agrupamiento de documentos de texto parecidos por su contenido, utilizando minería de texto sobre un dataset elegido (Base de datos de Gutenberg). Se utilizó el algoritmo TF-IDF para hallar similitud y el algoritmo K-means para el agrupamiento final. Además se analiza las diferencias entre los tiempos de ejecución de una implementación paralela en un ambiente HPC y este proyecto.

Palabras clave: *Big data, Minería de texto, Clustering, K-Means, Spark, Hadoop, Mapreduce, ETL, HPC (High Performance Computing), Machine learning.*

1. Introducción

El agrupamiento de documentos (Clustering[1]) se enfoca en relacionar ciertos documentos con otros parecidos basándose en alguna métrica que permita establecer la similaridad entre estos. Esto es muy útil cuando se desea dar a un lector documentos que debido a su similaridad puedan ser de su interés. Para realizar dicho agrupamiento es necesario utilizar la minería de texto, la cual se trata de analizar una colección de archivos de texto, con el objetivo de "extraer" las palabras clave, temas y relaciones entre los documentos; y descubrir tendencias a partir de las características globales, en lugar de hacerlo a partir de la semántica. Para el desarrollo de esta implementación se tomó como base el artículo "Similarity Measures for Text Document Clustering" [Huang, Anna.]. Lo que se propone es realizar el agrupamiento de los documentos de un dataset dado empleando técnicas de minería de texto, así mismo como utilizar bibliotecas de spark y el modelo ETL para el análisis y diseño del algoritmo. El desarrollo de este artículo comenzará explicando los conceptos básicos para la comprensión del artículo, continuando con el análisis y el diseño del algoritmo a implementar. Luego, en la explicación sobre la implementación, se ampliará con detalle la manera en que se realizó el mismo. Para finalizar, se mostrarán los resultados y serán analizados para comparar los tiempos finales que se obtuvieron del algoritmo implementado en HPC y el código desarrollado para esta práctica.

[1] tarea de agrupar un conjunto de objetos de tal manera que los miembros del mismo grupo (llamado clúster) sean más similares, en algún sentido u otro

2. Marco Teórico

Dado que este trabajo está enfocado en la minería de texto, es necesario primero plantear conceptualmente qué es minería de texto y sus diferencias con la minería de datos. La minería de texto - text mining - se trata de la estructuración de archivos, documentos, textos o registros de una colección, también conocida como dataset; a partir de la extracción de palabras clave, conceptos o ideas clave, temas y contenidos para su análisis posterior [Berry, Michael W., and Jacob Kogan, eds]. No debe confundirse con la minería de datos - data mining - que abarca un conjunto de herramientas y metodologías diseñadas para encontrar patrones interesantes y adquirir conocimiento a partir de la información contenida en las bases de datos, "the art of extracting information from data" [Tufféry, Stéphane.]. Por el contrario de la minería de datos, la minería de texto se interesa por la extracción de información global útil entre los documentos. Encontrar conexiones, tendencias y la relación entre los archivos que permitan la agrupación - clustering - en un número de categorías apropiada [Tonkin, Emma, and Gregory JL Tourte]. Los temas que se tratan principalmente en la minería de textos son: extracción de palabras clave, clasificación y agrupamiento, anomalías y detección de tendencias y flujos de texto. Por otra parte, la computación paralela se conoce como el uso simultáneo de los múltiples recursos de computación para solucionar un problema, el cual se divide en pequeñas tareas que se pueden resolver concurrentemente. cada división se descompone en una serie de instrucciones, que son ejecutadas en los diferentes procesadores.

También es importante hablar de Big Data. Big Data es un término evolutivo que describe cualquier cantidad voluminosa de datos estructurados, semiestructurados y no estructurados que tienen el potencial de ser extraídos para obtener información. Los datos grandes se caracterizan a menudo por tres Vs: el Volumen extremo de datos, la gran Variedad de tipos de datos y la Velocidad a la que se deben procesar los datos. Aunque los grandes datos no equivalen a ningún volumen específico de datos, el término se utiliza a menudo para describir terabytes, petabytes e incluso exabytes de datos capturados con el tiempo.

MapReduce es una técnica de procesamiento y un modelo de programación para computación distribuida. Se divide en 2 fases principales. La primera es MAP, que toma un set de datos almacenados en HDFS, recibe los datos línea por línea y los manipula de manera individual para convertirlos en una tupla <clave, valor>, creando así un nuevo set de datos. La segunda fase es REDUCE que toma la salida del paso anterior y convierte el grupo de tuplas, en un conjunto más pequeño a partir del parámetro clave.

Otro término relevante para la realización de esta práctica es Apache Spark[2], el cual es un framework open source para el procesamiento de datos masivos diseñado con tres prioridades en mente: velocidad, facilidad de uso, y capacidades avanzadas de analítica. Spark está cobrando gran popularidad porque viene a resolver varias de las limitaciones inherentes de Hadoop y MapReduce. Spark puede utilizarse junto co n Hadoop, pero no es requisito. Spark extiende el modelo MapReduce para hacerlo más rápido y habilitar más escenarios de análisis, como por ejemplo queries interactivos y procesamiento de flujos en tiempo real. Esto es posible ya que Spark usa un cluster de cómputo en memoria (in-memory)[Pedro Galvan]. .

Por último, cabe resaltar el concepto de "Machine Learning", el cual es el subcampo de las ciencias de la computación y una rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan a las computadoras aprender. De forma más concreta, se trata de crear programas capaces de generalizar comportamientos a partir de una información suministrada en forma de ejemplos.

[2] https://spark.apache.org/

3. Análisis y Diseño mediante Analitica de datos

3.1. ETL

Extract, Transform and Load («extraer, transformar y cargar», frecuentemente abreviado ETL) es el proceso que permite a las organizaciones mover datos desde múltiples fuentes, reformatearlos y limpiarlos, y cargarlos en otra base de datos, data mart, o data warehouse, o en otro sistema operacional para apoyar un proceso de negocio.

3.1.1. Extract: Extraer

La primera parte del proceso ETL consiste en extraer los datos desde los sistemas de origen. La mayoría de los proyectos de almacenamiento de datos fusionan datos provenientes de diferentes sistema. Cada sistema separado puede usar una organización diferente de los datos o formatos distintos. Los formatos de las fuentes normalmente se encuentran en bases de datos relacionales o ficheros planos, pero pueden incluir bases de datos no relacionales u otras estructuras diferentes. La extracción convierte los datos a un formato preparado para iniciar el proceso de transformación.

3.1.2. Transform: Transformar

La fase de transformación aplica una serie de reglas de negocio o funciones sobre los datos extraídos para convertirlos en datos que serán cargados. Algunas fuentes de datos requerirán alguna pequeña manipulación de los datos.

3.1.3. Load: Cargar

La fase de carga es el momento en el cual los datos de la fase anterior (transformación) son cargados en el sistema destino. Dependiendo de los requerimientos de la organización, este proceso puede abarcar una amplia variedad de acciones diferentes.

Se implementó esta metodología debido a que se tenía un dataset del cual era necesario extraer información importante además de transformarla en los tipos de datos requeridos para proceder a la carga del modelo. Este último fue utilizado en el procesamiento para obtener el resultado final de donde se sacaron las conclusiones de la práctica.

3.2. Algoritmos y estructuras de datos

Apache Spark combina un sistema de computación distribuida a través de clusters con una manera sencilla de escribir programas. Spark surgió después de MapReduce, mejorando la manera como se trabaja con grandes conjuntos de datos, y que permite la ejecución paralela en un gran número de máquinas simultáneamente. La arquitectura de MapReduce provee una escalabilidad horizontal si es necesario agregar más máquinas a medida que incrementa la cantidad de datos. Spark mantiene la escalabilidad lineal y la tolerancia a fallos de MapReduce, pero agrega nuevas herramientas para su uso tales como DAG y RDD.

3.2.1. DAG (Grafo Acíclico Dirigido)

Este se refiere a un grafo dirigido que no tiene ciclos, es decir, no hay un camino directo desde un nodo hasta el mismo. Spark soporta el flujo de datos acíclico. Cada tarea de Spark crea un DAG de etapas de trabajo para que se ejecuten en un determinado cluster. Los grafos DAG creados por Spark pueden tener cualquier número de etapas. Spark con DAG es más rápido que MapReduce por el hecho de que no tiene que escribir en disco los resultados obtenidos en las etapas intermedias del grafo. MapReduce, sin embargo, debe escribir en disco los resultados entre las etapas Map y Reduce.

3.2.2. RDD (Resilient Distributed Dataset)

Spark mejora con respecto a los demás sistemas en cuanto a la computación en memoria. RDD permite realizar operaciones sobre grandes cantidades de datos en clusters de una manera rápida y tolerante a fallos. Una vez que los datos han sido leídos como objetos RDD en Spark, pueden realizarse diversas operaciones

mediante sus APIs. Los dos tipos de operaciones que se pueden realizar son transformaciones (tras aplicar una transformación, obtenemos un nuevo y modificado RDD basado en el original) y acciones (una acción consiste simplemente en aplicar una operación sobre un RDD y obtener un valor como resultado, que dependerá del tipo de operación). Dado que las tareas de Spark pueden necesitar realizar diversas acciones o transformaciones sobre un conjunto de datos en particular, es altamente recomendable y beneficioso en cuanto a eficiencia el almacenar RDDs en memoria para un rápido acceso a los mismos.

3.2.3. Estrategia de implementación

Como se puede observar en la Figura 1, las líneas rojas representan transformación y las verdes operación.

A partir de una variable de entorno llamada context se crea un objeto RDD llamado "archivos" en donde se lee de la ruta especificada el dataset a procesar.

Una vez creado el RDD inicial se realizan transformaciones para crear más objetos RDD a partir del primero. Dichas transformaciones se expresan en términos de programación funcional y no eliminan el RDD original, sino que crean uno nuevo. Las transformaciones que se hacen son:

$$nombreArchivos = archivos.keys().collect()$$

$$docs = archivos.values().map(lambda documento : documento.split(""))$$

$$tf = hashingTF.transform(docs)$$

$$idf = IDF(minDocFreq = 2).fit(tf)$$

$$tfidf = idf.transform(tf)$$

El RDD resultante de las transformaciones será luego procesado por el KMeans, con un determinado k y un número máximo de iteraciones. Esto así:

$$clusters = KMeans.train(tfidf, k, maxIterations = maxIt)$$

Tras realizar las acciones y transformaciones necesarias sobre los datos, los objetos RDD

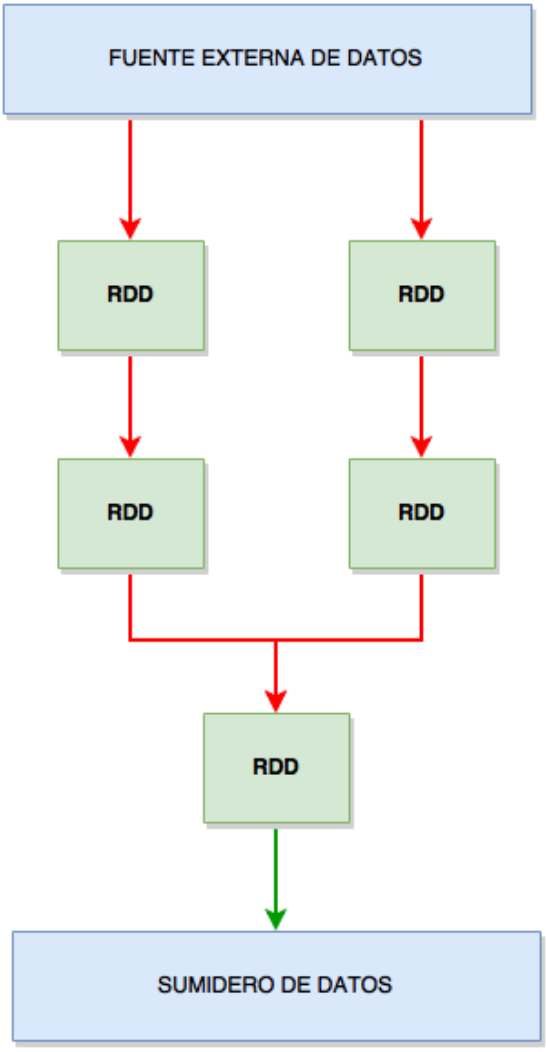

Figura 1: *Grafo de ejecución*

convergieron para crear el RDD final. Este RDD puede ser almacenado de la siguiente manera:

$$clusterid = clusters.predict(tfidf).collect()$$

$$dictionary = dict(zip(nombreArchivos, clusterid))$$

4. Implementación

La implementación del algoritmo se desarrolló principalmente con el uso de PySpark[3], que es un API de Apache Spark para Python[4]. De PySpark se utilizó la biblioteca mllib, que interactúa con las biblioteca Numpy[5] y R[6], que incluye algoritmos de clustering como k-means y TF-IDF.

4.1. TF-IDF

Term Frequency - Inverse Document Frequency (TF-IDF) es un método de vectorización ampliamente utilizado en la minería de texto para reflejar la importancia de un término en un documento perteneciente a un dataset. El algoritmo TF-IDF se compone de dos partes: TF e IDF

- TF: Es el número de veces que aparece un término en un documento.
- IDF: Primero se necesita calcular DF (Document Frequency), el cual es el número de documentos que contienen cierto término. IDF (Inverse Document Frequency) es una medida numérica de cuánta información posee un término, es decir, si un término se repite constantemente en los documentos, se vuelve una palabra irrelevante, porque la información que trae no es especial.

El producto de estas dos medidas genera como resultado una medida mucho más precisa y confiable que posteriormente nos dará información de la similaridad que puedan tener los documentos analizados. Si sólo se utilizará el TF como medida de importancia, se podrían tener en cuenta muchos términos los cuales no son relevantes, por ejemplo los StopWords.

4.2. K-Means

Como algoritmo de agrupamiento, se optó por utilizar K-means, esto ya que es uno de los algoritmos más utilizados para esta modalidad.

Debido a que es un algoritmo no supervisado que resuelve el problema de clustering clasificando los datos en K clusters de una manera simple y fácil. En resumen, el algoritmo de K-means funciona de la siguiente manera:

1. Se ubican los k centroides aleatoriamente en el espacio de los documentos, cada centroide representará el centro de un grupo de documentos que tienen características similares.

2. Se asigna cada documento al grupo del centroide más cercano de este.

3. Cuando todos los documentos son asignados, se recalcula la posición de los centroides, calculando el promedio de las posiciones de los documentos de su propio grupo.

4. Se repite el paso 2 y 3, ya sea con un número fijo de iteraciones o hasta que los centroides no cambien de posición al realizar el paso 3.

La implementación en spark.mllib recibe los siguientes parámetros:

- k: es el número deseado de clusters.
- maxIterations: es el máximo número de iteraciones para ejecutar.
- initializationMode: Modo de ejecución, este es paralelo por defecto.
- runs: este parámetro no tiene efecto en las nuevas versiones de spark.
- initializationSteps: determina el número de paso en el algoritmo k means paralelo.
- epsilon: determina la distancia que se considera que el algoritmo debe converger.

[3] http://spark.apache.org/docs/latest/api/python/index.html
[4] https://www.python.org/
[5] http://www.numpy.org/
[6] https://www.r-project.org/

HPC			
		K	
		2	5
Archivos en dataset	6	1,2	1,3
	20	2,0	2,8
	90	12,4	10,2
	151	15,2	11,6
	461	26,8	28,7

Figura 2: *Tiempos de ejecucion en el ambiente HPC.*

BIG DATA			
		K	
		2	5
Archivos en dataset	6	35,0	40,0
	20	37,0	44,0
	90	50,0	60,0
	151	55,0	81,0
	461	104,0	112,0

Figura 3: *Tiempos de ejecucion en el ambiente Big Data.*

5. Análisis de resultados

HPC utiliza muchos más nodos y gránulos finos lo que le permite trabajar con muchos más datos al mismo tiempo, con lo cual logra reducir los tiempos notablemente. Por el contrario, big data utiliza gránulos más gruesos y menos nodos donde su principal preocupación es la cantidad de datos con los cuales puede trabajar.

Las bibliotecas utilizadas para desarrollar el proyecto de big data son más centradas en la cantidad de datos que pueden procesar, es decir su factor fundamental es el volumen de datos. En cambio, los entornos de computación de alto rendimiento, logran poner su foco en el tiempo de ejecución, lo que quiere decir que su principal preocupación son los tiempos con los que entregan la 'solución'. Por este motivo, se puede decir que la teoría concuerda con los tiempos obtenidos.

Para comparar los resultados de HPC y Big Data, expresados en la Figura 4 es necesario tener en cuenta que la principal caracteristica de HPC es su gran capacidad de computo, mientras que la de Big Data es principalmente su gran capacidad de almacenamiento. Esto se evidencia en los dos ambientes de ejecución, en el cual HPC utilizaba 50 nucleos de procesamiento mientras que Big Data en no mas de 8.

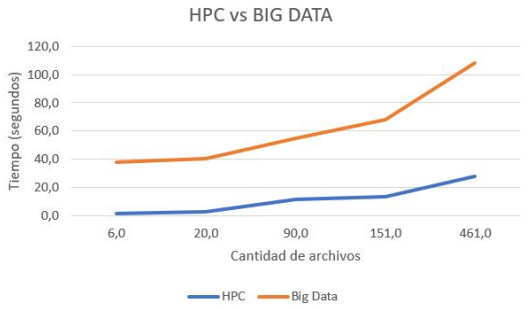

Figura 4: *Grafica de comparación entre los tiempos de ejecucion en ambientes HPC y Big Data.*

6. Conclusiones

- Big Data es utilizado principalmente para casos en los que se tienen grandes cantidades de datos que no son posibles de procesar en una sola máquina, por lo que se requiere distribuir estos datos para poder obtener un buen resultado.
- Big Data se enfoca más en abarcar mayor cantidad de datos a procesar y HPC se enfoca en obtener mayores velocidades de procesamiento.
- En esta práctica se observó que HPC procesa a mayor velocidad, pero puede ocurrir que este se quede corto a la hora de procesar datos que no puedan ser almacenados en una sola máquina, lo cual si permite Big Data que aunque no es tan rápido, si permite el procesamiento de estas cantidades de datos.
- Se utilizó adecuadamente la metodología para el diseño de algoritmos ETL.

- Se aplicaron las tecnologías y modelos de programación en Big Data.
- Se analizaron los resultados entre información sobre una programación en paralelo y las tecnologías existentes para big data.
- Se entendieron las limitaciones tanto para HPC como para big data concluyendo cuándo y en qué ambientes era mejor una cosa que otra.

Referencias

[Huang, Anna.] Similarity measures for text document clustering. Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008) *Christchurch, New Zealand*, 2008.

[Tufféry, Stéphane.] Data mining and statistics for decision making. *Vol. 2. Chichester*, Wiley, 2011.

[Berry, Michael W., and Jacob Kogan, eds] Text mining: applications and theory *John Wiley and Sons*, 2010.

[Tonkin, Emma, and Gregory JL Tourte] Working with Text: Tools, Techniques and Approaches for Text Mining *Elsevier*, 2016.

[Pedro Galvan] Un Vistazo a Apache Spark Streaming *SG Buzz*, 2016.